2015/2016

Relatório de Trabalho de P2

Discentes:

- Daniel Murraças 32371

- Gil Catarino 32378

Docentes:

- Salvador Abreu

- Lígia Ferreira

Introdução

Nesta cadeira foi-nos pedido para implementar um jogo em java chamado **SMOTHY**, que é um jogo baseado noutro conhecido (o “Candy Crush). O principal objectivo é remover as cores apresentadas nu tabuleiro até que não haja mais nenhuma apresentada no tabuleiro. Para que estas sejam removidas, a cor seleccionada através de inputs para o utilizador, tem que ter pelo menos uma adjacente igual à cor indicada. O jogo termina assim que não existir mais cores, ou não haver mais jogadas possíveis, apresentado o respectivo resultado.

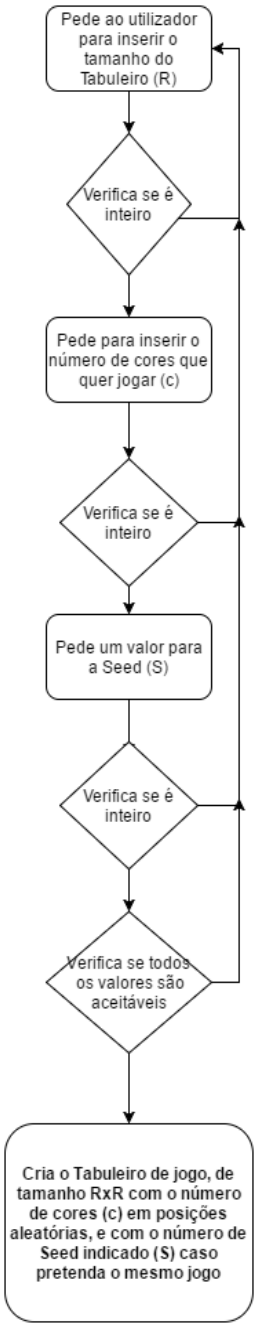
Começámos por criar uma função que cria um tabuleiro (uma matriz), que vai receber arrays, onde esses arrays irão ter o valor/nome de uma cor. Como argumentos, essa função precisa do tamanho do tabuleiro, o número de cores desejado para jogar, e o valor da seed para depois colocar essas cores numa forma aleatória através da classe Random do java.

```
public void  criartabuleiro(int S,int n, int c){



cores = new String[5];
cores[0] = "Blue";
cores[1] = "Red";
cores[2] = "Green";
cores[3]= "Black";
cores[4]= "Beige";

matriz = new String [n+2][n+2];


    Random r = new Random(S);
    for(int linha=1 ; linha <= n ; linha++){
        for(int coluna = 1; coluna <= n ; coluna ++){
           int a=r.nextInt(c);
           matriz[linha][coluna]=cores[a];

        }
    }
}
```

Também teve que se limitar essa mesma matriz com nulls para se poder trabalhar (mover, alterar valores) só ao que estava dentro dessa mesma matriz.

Ao mesmo tempo criava-se uma função para começar a fazer os inputs ao utilizador, para obter através da utilização de Scanner os valores do tamanho do tabuleiro, o nº de cores, e seed, e as coordenadas a indicar (por linhas e colunas), onde através de Try e Catch verifica se as condições são aceitáveis para o programa.

```
public void inputs(int n){
        Scanner sc = new Scanner(System.in);
        tospace(n);
        baixar(n);
        verificaColuna(n);
        imprime(n);
        tonull(n);
        if (verificaprimeiro(n)==true){
            try{



                System.out.println("Numero da Linha:");
                int x = sc.nextInt();
                if(x>n || x<=0){System.out.println("Insira uma linha menor ou igual que " + n + " e maior  ou igual a 1 para jogar sff");
                    loopcord(n);}


                System.out.println("Numero da Coluna:");
                int y = sc.nextInt();
                if(y>n || y<=0){System.out.println("Insira uma coluna menor ou igual que " + n + " e maior  ou igual a 1 para jogar sff");
                loopcord(n);}


                if (checkfirst(n,x,y) == true){
                    construc(n,x,y);
                }
            }
            catch(InputMismatchException e){
                System.out.println("Por favor, insira numeros");
        //      sc.next();
            }
        }
}
```



Depois de ter os valores necessários, imprime o tabuleiro no ecrã, onde pede as coordenadas das linhas e colunas, e se forem dentro dos valores da matriz (1 a R), verifica o que tem nessa coordenada, e consoante o que seja (espaço em branco, uma cor isolada, ou uma cor com adjacentes compatíveis), executa da melhor forma para o programa continuar a correr.

Isto é o que a função “checkfirst” e “check” fazem. Através da coordenada indicada, procuram os seus adjacentes. Caso tenha, chama a função “removenull” que remove essas cores todas que forem adjacentes, dentro do limite da matriz inicial.

```
public void removenull(int n , int x, int y, String palavra){
        for(int linha=1 ; linha <= n ; linha++){
    for(int coluna = 1; coluna <= n ; coluna ++){

        if (linha+1 <=n) {
            if (matriz[linha+1][coluna]!= null ){
            if (matriz[linha][coluna].equals("    ") && matriz[linha+1][coluna].equals(palavra)) {
                    check(n,linha+1,coluna);

            }
            }
        }
       if(coluna+1 <= n) {
            if (matriz[linha][coluna+1]!= null ){
            if (matriz[linha][coluna].equals("    ") && matriz[linha][coluna+1].equals(palavra)) {
                    check(n,linha,coluna+1);

            }
            }
        }
        if(linha-1 >=1) {
```

```
public String check(int n,int x, int y){

    String palavra = "";

            int x1 = x+1;
            int y1 = y+1;
            int x2 = x-1;
            int y2 = y-1;
            palavra= matriz [x][y];  // Minha palavra


           remover(n, x, y);
            if(x1 < n+2) {
                if(y1 < n+2) {
                    if(x2 > -1) {
                        if(y2 > -1) {

                                /* */
                                removenull(n,x,y,palavra);

                        }
                       }
                    }

            }
```

A função “remover” simplesmente passa o que estava na coordenada indicada para um espaço em branco “ “.

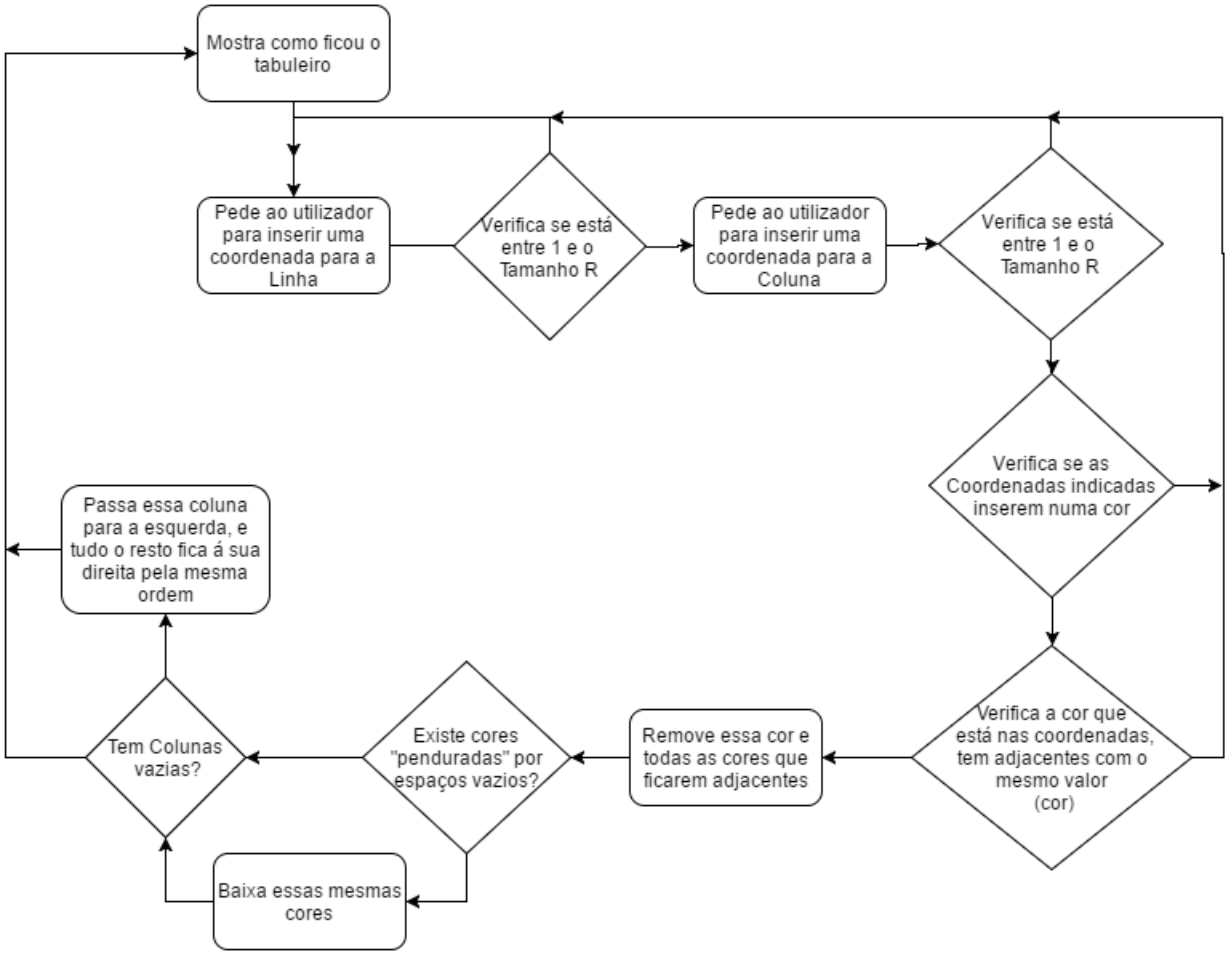
Depois disso foi necessário percorrer a matriz e encontrar as cores que sobravam separadas na mesma coluna com espaços vazios, e baixa-las (trocando-as de posição com o espaço vazio) de modo que todas as cores estivessem o mais próximo do “solo” do tabuleiro. Para isso criou-se o método “baixar” que faz exactamente isso.

```
public void baixar(int n){
        String espaco ="    ";

        while(verificabaixo(n)){

            for(int linha=1 ; linha <= n ; linha++){
                for(int coluna = 1; coluna <= n ; coluna ++){

                    if(matriz[linha][coluna].equals(espaco)){
                        if (matriz[linha-1][coluna]!=null){
                            matriz[linha][coluna]= matriz[linha-1][coluna];
                            matriz[linha-1][coluna]="    ";
                        }

                    }
                }
            }
        }
}
```

Logo a seguir, tinha que se verificar se existia alguma coluna completamente vazia (sem cores). Se existisse, teria que ficar à esquerda, de modo que as colunas com valores ficariam à sua direita pela mesma ordem. Assim sendo, foi criado entre outros, o método “verificaColuna”.

```
public void verificaColuna(int n){
   String espaco ="    ";
       for(int linha=1 ; linha <= n ; linha++){
           for(int coluna = 1; coluna <= n ; coluna ++){


               for (int i = 1; i <=n; i++){
                   if(matriz[n][i-1]!= null){
                       if (matriz[n][i].equals("    ")){

                           for( linha =1; linha <= n; linha++){
                                 matriz[linha][i]= matriz[linha][i-1];
                                 matriz[linha][i-1] = espaco;
                                   if(matriz[linha][i-1]== null){
                                       break;
                                   }
                           }
                       }
                   }

               }
           }
       }

}
```

Para acabar, foi criar um loop, para que este processo acontecesse sempre, só ir alterando os resultados no tabuleiro, que iria ser apresentado com as alterações na próxima jogada. Assim que já não houvesse mais jogadas possíveis, o programa parava mostrando o resultado (através do método resultado que calcula a pontuação) obtido pelas jogadas e felicitando o utilizador pelo sucedido.



Conclusão

Neste trabalho realizamos um jogo, que era do estilo, candy crush, chamode de smoothy.

Cumprimos todos os objectivos que nos tinham proposto para a realização do trabalho, dentro do prazo pretendido, fazendo com que o jogo funcione correctamente.

Este trabalho foi muito importante para o nosso conhecimento e compreensão, pois permitiu-nos ficar com melhor conhecimento sobre a programação java, e como a implementar.